
# SAME GAME

RAPPORT DE LA SAE21_2025
MATHIAS ET LUKAS TP06

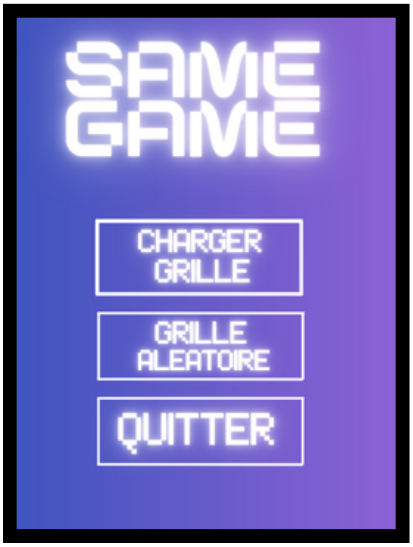

# SOMMAIRE

# 1. Introduction : Same Game

Le programme a pour but de permettre à l'utilisateur de jouer à une partie de “Same Game”. Dans ce jeu, une grille de dix lignes et quinze colonnes est présentée au joueur. Chacune des cases de cette grille contient un bloc de couleur : ce bloc peut être Rouge, Vert ou Bleu. L’utilisateur peut choisir de définir cette grille par un fichier, ou alors de manière aléatoire. Dans la grille de jeu, lorsqu’au moins deux blocs de même couleur sont adjacents, il font partie d’un même groupe. Le but du jeu est de détruire ces groupes de couleurs, en essayant de faire en sorte de supprimer le plus de blocs possibles en une seule suppression, afin de maximiser son nombre de points. L’entièreté du jeu est jouable à la souris : si l’utilisateur survole un groupe, celui-ci est visuellement accentué. Si ledit groupe n’est plus survolé, il n’est plus visuellement accentué. Un groupe est supprimable en cliquant sur l’un de ses blocs.
Le jeu s'arrête lorsque il n’existe plus aucun groupe à supprimer. Dans ce cas, l’utilisateur peut voir son score ou quitter le jeu.

# 2. Fonctionnalités du programme

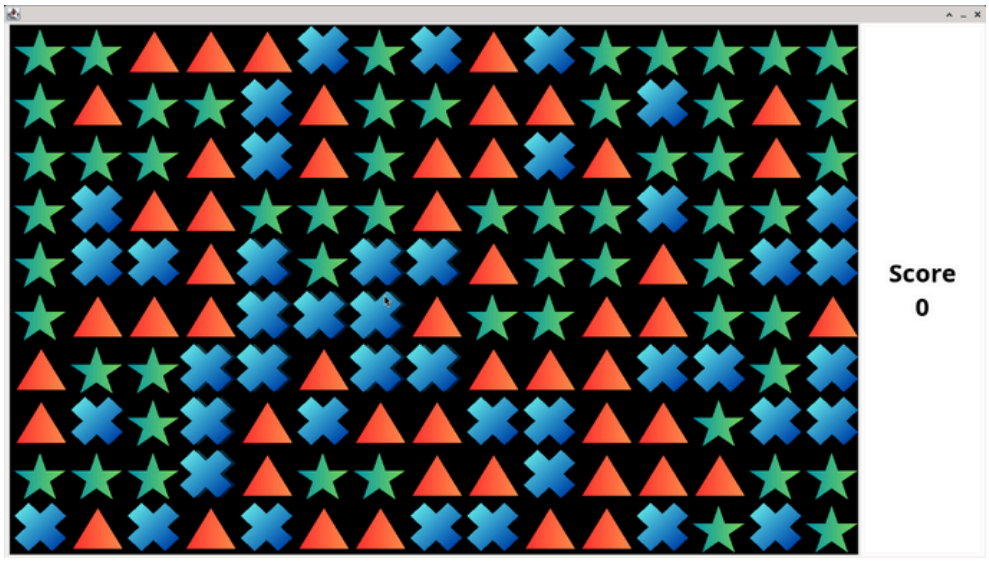
- Menu principal : Possibilité de lancer une partie ou de quitter le jeu (fermer le programme)
- Présence d'un sous-menu qui permet à l'utilisateur de :
  - Lancer une partie avec une grille chargée via un fichier
  - Lancer une partie avec une grille chargée aléatoirement
  - Quitter le jeu (fermer le programme)

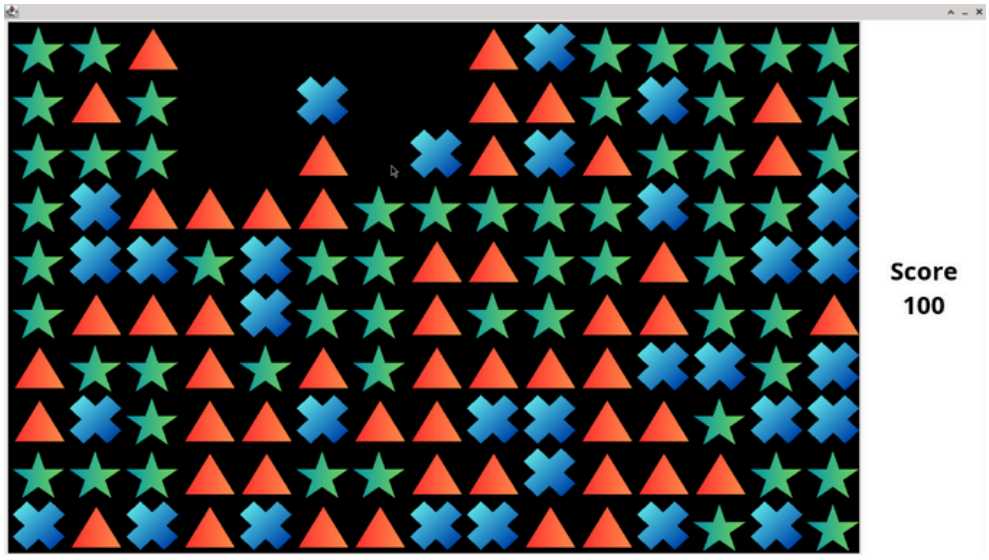

- Lorsque l'utilisateur décide de lancer une partie avec la première option, un JFileChooser lui permet de choisir le fichier avec lequel la grille sera chargée.
- S'il choisit la deuxième option, la grille de jeu sera totalement aléatoire.
- Une fois la grille chargée, la fenêtre affiche la grille ainsi que le score du joueur à droite.
- Lorsque l'utilisateur survole un groupe de même couleur, le sprite des blocs concernés change afin de lui signaler l'ensemble des blocs appartenant au même groupe que le bloc qu'il survole.
- Lorsque l'utilisateur ne survole plus un groupe de même couleur, les blocs concernés retrouvent leur sprite original.
- Lorsque l'utilisateur clique sur un groupe de même couleur, le groupe disparaît, puis les blocs flottants "tombent" afin de créer un effet de gravité.
- A chaque suppression, si une colonne non vide se situe à droite d'une colonne vide, on décale les colonnes à droite d'un cran vers la gauche.
- A chaque suppression, le score du joueur, affiché à droite, s'actualise automatiquement en fonction du nombre de blocs qu'il a détruits.

Exemple d'une partie :

- Tour 1




- Tour 2

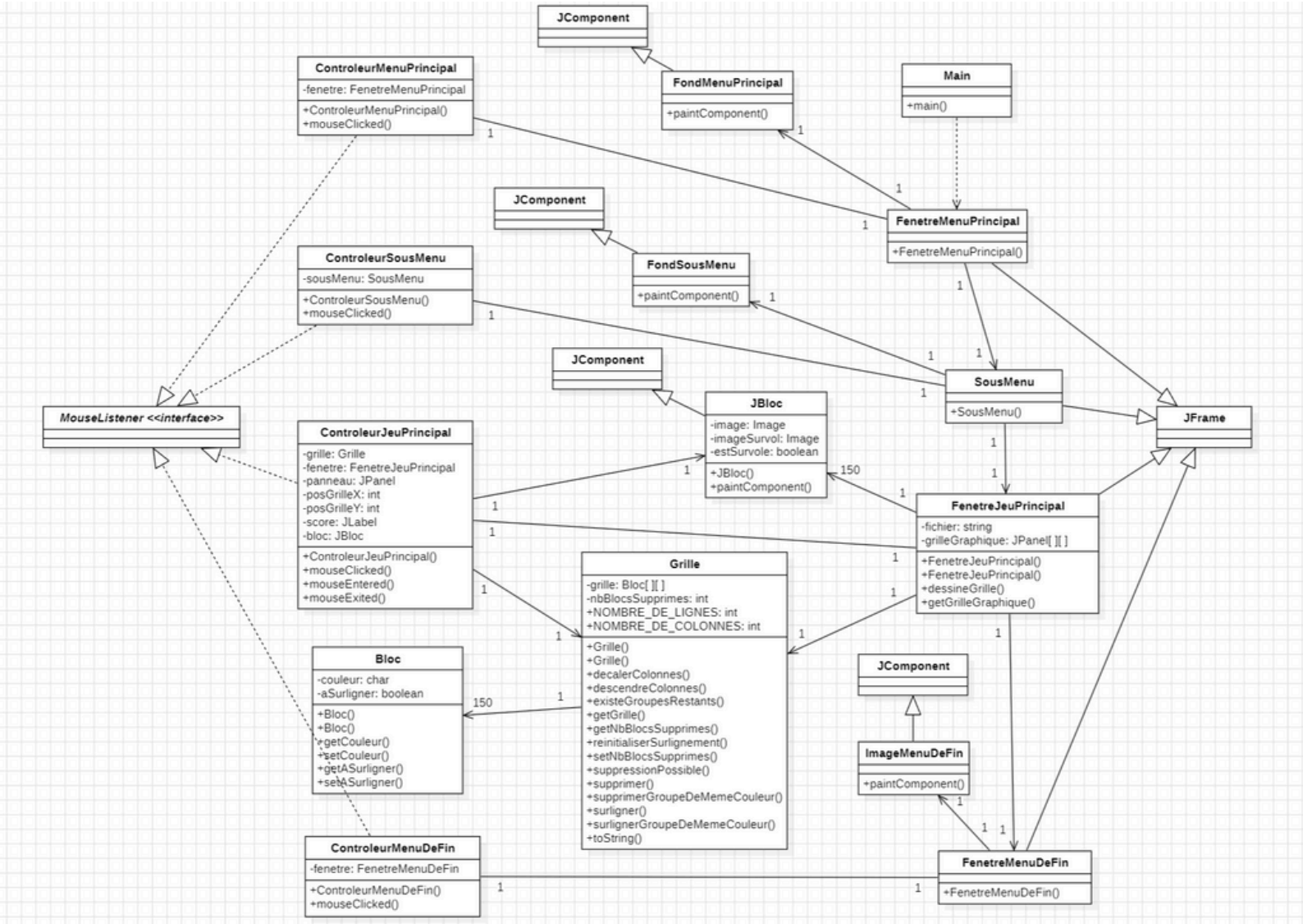



- Menu de fin :
  - Permet de voir le score avant de quitter le jeu
  - Permet de quitter directement le jeu (fermer le programme)

# 3. Structure du programme (Diagramme)

Dans l'ensemble, la structure du programme cherche à respecter au maximum le motif Modèle-Vue-Contrôleur, afin de garantir une certaine forme de rétrocompatibilité.

- Modèle : Bloc.java Grille.java
- Vue : FenetreMenuPrincipal.java FenetreMenuDeFin.java etc.
- Contrôleur : ControleurMenuPrincipal.java ControleurSousMenu.java etc.



# 4. Exposition de l'algorithme qui identifie les groupes

L'algorithme d'identification de groupes de même couleur se déroule ainsi :

- On marque la case sélectionnée comme étant en surbrillance (de couleur "S") ← Garantit l'arrêt
- On calcule les coordonnées de ses quatre voisins
- Pour chacun des quatre voisins :
  - Si sa couleur est identique à celle du bloc initial :
    - On rappelle la méthode récursivement en désignant ce voisin comme le nouveau "bloc sélectionné"

Voici ce qu'il donne en pseudo-code :

```
PROCEDURE identifierGroupes(int x, int y, char couleur)
    couleurCaseActuelle <- la couleur du bloc (x, y)
    couleur du bloc (x, y) <- 'S'

    SI voisin du haut est de même couleur que couleurCaseActuelle:
        identifierGroupes(x, y-1, couleur)
    FIN SI

    SI voisin du bas est de même couleur que couleurCaseActuelle:
        identifierGroupes(x, y+1, couleur)
    FIN SI

    SI voisin de gauche est de même couleur que couleurCaseActuelle:
        identifierGroupes(x-1, y, couleur)
    FIN SI

    SI voisin de droite est de même couleur que couleurCaseActuelle:
        identifierGroupes(x+1, y, couleur)
    FIN SI
FIN PROCEDURE
```

# 5. Conclusion Mathias

“

Ce projet, pour ma part, a été globalement une réussite. Nous avons répondu à ce qui était demandé.
Je me suis beaucoup occupé de la partie front-end, notamment des boutons des différents menus et de la gestion du fond. J’ai trouvé que nous avions réussi à mieux nous organiser que lors de la première SAE, ce qui nous a permis de faire des choses en plus, comme un design correct pour rendre le jeu plus agréable.
Réaliser cette SAE m’a vraiment permis de comprendre certaines fonctions que je ne maîtrisais pas bien, comme MouseEvent.
Mon camarade s’est chargé de la majorité de la partie back-end et m’a expliqué beaucoup de choses sur la façon de créer la grille. J’ai pu suivre le rythme et mener à bien cette SAE grâce à mon binôme, qui est vraiment fort pour expliquer et appliquer ses connaissances de manière rigoureuse.
J’ai beaucoup aimé faire le Makefile, ainsi que travailler sur les différents menus, leurs fonctionnalités et la direction artistique.
Par rapport à la SAE précédente, j’ai trouvé que notre communication était plus claire, et que la répartition des tâches selon nos compétences a vraiment optimisé notre travail.
Ce projet m’a permis de développer mes compétences en Java, mais aussi de mieux collaborer en équipe et de trouver des solutions ensemble.

# 6. Conclusion Lukas

“

Le projet s’est globalement très bien déroulé, toutes les fonctionnalités demandées sont présentes et je trouve que le jeu est très plaisant à regarder.
Ayant majoritairement travaillé sur la partie back-end, ce projet m’a permis de me perfectionner dans mon utilisation de la programmation orientée objet. Le choix des différentes classes et de leurs membres demandait une certaine réflexion, et j’ai passé beaucoup de temps à envisager la structure globale du programme avec Mathias avant de commencer à coder quoique ce soit. Je pense que cette démarche a grandement facilité le déroulement du projet, le rendant plus facile à mettre à jour, à la fois pour moi et pour Mathias. Par ailleurs, la nécessité de faire une Javadoc m’a permis d’apprendre à commenter mon code au fur et à mesure, et ce de façon plus claire, en détaillant la signature de chaque méthode (chose que je ne faisais pas forcément avant).
La gestion des affichages, le choix des couleurs, des polices et des boutons a été fait par mon binôme. Pour ma part je n’ai fait que les sprites représentant les blocs de la grille. J’apprécie beaucoup l’esthétique globale du projet, je la trouve très réussie - je me dois de souligner qu’elle le serait beaucoup moins sans Mathias.
De la même manière que nôtre projet précédent, nous avons su dégager une forme de complémentarité en mettant à profit nos compétences respectives afin d’arriver à un résultat satisfaisant. J’aurais aimé ajouter quelques fonctionnalités supplémentaires, comme le stockage du Meilleur Score dans un fichier, ainsi que son affichage, mais nous avons jugé qu’il était plus judicieux de concentrer notre temps sur ce qui nous était demandé.
Dans l’ensemble, ce projet s’est montré très enrichissant et m’a permis de perfectionner mes connaissances en Java ainsi que ma façon de collaborer en équipe.

# Fin de rapport